Un problème de vue

```
public static void main(String[] args) throws Exception {
  A a = new A() ;              // Création d'un objet a de la classe A
  a.start() ;                  // Lancement du thread a
  a.valeur = 1 ;               // Modification de l'attribut valeur
  a.fin = true ;               // Modification de l'attribut fin
}

static class A extends Thread {
  public int valeur = 0 ;
  public boolean fin = false ;

  public void run() {
    while(! fin) {} ;          // Attente active
    System.out.println(valeur);
  }
}
```

Ce programme termine-t-il ? Peut-il afficher 0 ?

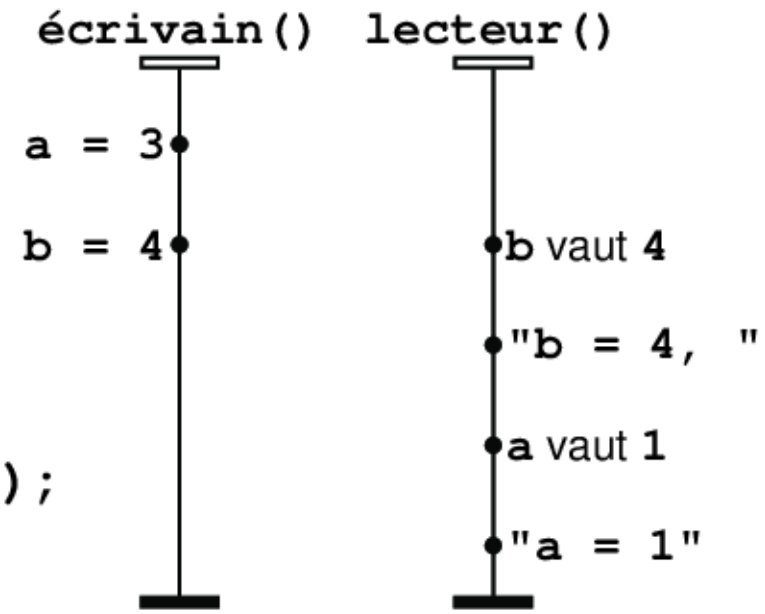
Ajout du mot-clef « volatile »

```
public static void main(String[] args) throws Exception {
  A a = new A() ;              // Création d'un objet a de la classe A
  a.start() ;                  // Lancement du thread a
  a.valeur = 1 ;               // Modification de l'attribut valeur
  a.fin = true ;               // Modification de l'attribut fin
}

static class A extends Thread {
  public int valeur = 0 ;
  public volatile boolean fin = false ;

  public void run() {
    while(! fin) {} ;          // Attente active
    System.out.println(valeur);
  }
}
```

Ce programme termine-t-il ? Peut-il afficher 0 ?

Une exécution (déjà vue) qui n'est pas séquentiellement consistante

```
class NotSoSimple {
  int a = 1, b = 2;
  void écrivain() {
    a = 3;
    b = 4;
  }
  void lecteur() {
    System.out.print("b = " + b +", ");
    System.out.println("a = " + a);
  }
}
```



Ce programme peut légalement afficher **`"b = 4, a = 1"`**. Mais comment ?

① La mise-à-jour de la valeur de **`a`** ne semble pas avoir été réalisée, *ou bien*

② Le programme ne semble pas s'exécuter dans l'ordre des instructions de **`écrivain()`**.

Garantie des programmes bien synchronisés

Bonne nouvelle :

≪ Le modèle mémoire Java garantit que les exécutions d'un programme **sans data-race** sont toutes **séquentiellement consistantes !** ≫

C'est bien ce que l'on veut !

Ce que ça veut dire en pratique.

Si la seule explication possible d'un résultat inattendu du programme est que

— les écritures n'ont pas été correctement réalisées en mémoire, ou

— les instructions du programme n'ont pas été réalisées dans l'ordre

c'est-à-dire que l'exécution observée n'est pas *séquentiellement consistante*, alors le programme contient **nécessairement** une *data-race*.

Ça peut aider pour débugger !

Code corrigé par un expert (mais avec encore une data-race)

```
class A {
  final int f;
  public A() { f = 42 ; }
}
class B {
  A a;
  static void écrivain() { a = new A() ; }
  static void lecteur() {
    A copie = a ;
    if ( copie != null ) println(copie.f) ;
  }
}
```

Le thread appliquant **`lecteur()`** pourra maintenant uniquement afficher :

- rien, s'il est trop rapide pour observer que l'objet **`a`** a été construit ;
- ou **`"42"`**, si le premier thread est suffisamment rapide pour construire **`a`**.

En résumé

Le modèle mémoire Java formalise les exécutions légales d'un programme Java et détermine à quel moment les modifications effectuées par un thread seront nécessairement *vues* par un autre thread.

Il définit en particulier la sémantique du mot-clef **`volatile`** ; celui-ci induit une visibilité, comme les verrous.

Certaines méthodes des threads : **`start()`**, **`join()`**, **`interrupt()`**, etc. ou des tâches : **`submit()`**, **`call()`**, **`get()`**, etc. garantissent également des synchronisations de la mémoire.

Enfin, puisque Java garantit la *consistance séquentielle* des exécutions sans data-race, tout programme inconsistant résulte nécessairement d'un défaut de synchronisation.